Discussion and correspondence
An alternative for the 'completer' function

W. A. Zaremba

Bechtel Inc, San Francisco, California

The need to provide conditional exits out of some large program units, such as loops or closed clauses, arises frequently in practice and is usually accomplished by jumps and labels. In this note we define an alternative mechanism avoiding this disadvantage at the cost of introducing one additional delimiter.

(Received June 1978; revised February 1980)

The need to provide conditional exits out of some large program units, such as loops or closed clauses, arises frequently in practice and is usually accomplished by jumps and labels, as in the following skeleton program in ALGOL 60:

for . . . **do**
begin *u*;*u*; . . . **if** . . . **then go to** *L*; . . . *u*;*u* **end**; *L*: . . .

or by the 'completer' construction of ALGOL 68:

BEGIN *u*;*u* . . . **IF** . . . **THEN** *L*; . . . **FI**; *u*;*u*; . . . **EXIT** *L*: *u*;*u*; . . . **END**

where *u* stands for some arbitrary computational unit, possibly another contained closed clause, and *L* is some label.

This type of flow control is contrary to the principles of structured programming and in this note we define an alternative mechanism avoiding this disadvantage at the cost of introducing one additional delimiter. The new construction is described in the context of ALGOL 68, but the results are applicable to other programming languages allowing the nested structuring, such as ALGOL 60, PL1, Pascal, etc.

Assume now that at the desired exit point we put a special form of conditional clause distinguished by the use of a new delimiter, say 'CUT' instead of 'THEN' (echoing film directors' language?), which opens the branch containing a series of units to be executed, *followed by a jump to the start of the first serial unit after the innermost closed clause enveloping the entire conditional.* In ALGOL 68 this would be some range contained between nesting delimiter pairs:

BEGIN . . . END or (. . .) or DO . . . OD or THEN . . . ELSE or ELSE . . . FI, etc.

Since exit is the last action in the 'cut' branch, no serial unit can be blocked within it, and since we only allow 'CUT' as a replacement of 'THEN' but not of 'ELSE' we can guarantee *syntactically* the existence of at least one path through the control tree, hence any serial units after the conditional remain accessible. Clearly, an IF clause not containing an ELSE branch also implicitly has such a through path of flow control, so that we are, in all cases, in line with ALGOL 68 philosophy expressed in the introduction to the revised report (van Wijngaarden *et al*, 1976) as: '. . . most syntactical and many other errors can be detected easily . . . . Furthermore, the opportunities for making such errors are greatly restricted.'

The restriction of 'CUT' to replace the 'THEN' branch only enforces a reasonable programming discipline, but the keyword itself is a kind of 'remote action token', just like DO, WHILE, etc. so that its effect comes at a later place in the code than the one at which it actually appears. It would be possible to avoid this separation of command and action by merging 'CUT' into the delimiters *closing* the exiting branch, e.g. CUTELSE, CUTELIF, CUTFI. Unfortunately this would not only multiply the number of new keywords, but also require a complicated syntax to prevent the possibility of blocked code. Therefore we conclude that this idea cannot be accepted.

It would be easy to generalise the idea of forward exiting jump by allowing also its *inversion*, i.e. a *backward jump to the beginning of the innermost range containing the entire conditional* (encoded possibly by, say, 'CUTBACK'). An obvious application would be in screening terminal entry data for errors, each one sending the user back to re-enter the input line. Any closed clause could now be made into a nest of loops, all starting at its beginning but reversing at different places in its range. Used within genuine loops however, this would cause them to restart upon each executed back transfer—hardly a sensible effect! Another possible extension would be to permit exits *through any number of levels of nested conditional clauses* stopping ultimately at the first unit following the innermost ordinary closed clause containing the nest. While this causes no blocking, we could not arrest the consecutive jumps at any

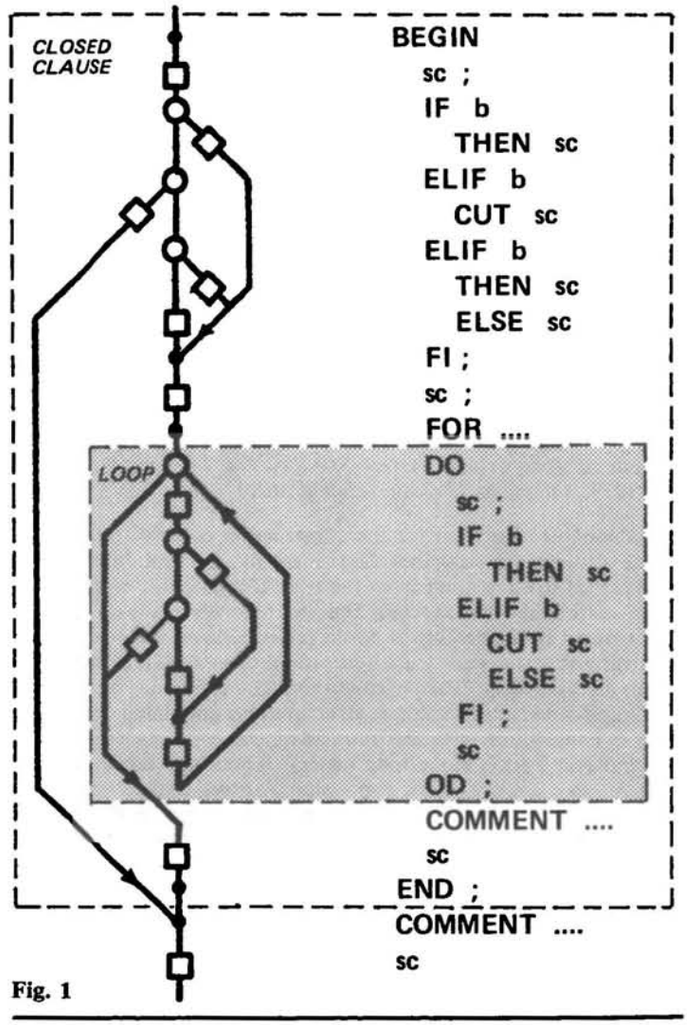

Fig. 1

intermediate layer without the introduction of another special keyword. Since it is so easy to produce undesirable side effects we will ignore both these possibilities in further discussion.

To define more formally the new construct we shall adopt a variant of regular expressions language with the following replicators used in the exponent position:

ϕ option : one or none,

+ sequence : one or more,

* sequence option: empty or a sequence.

Braces { } serve to delimit the scope of replicated construction, a choice is indicated by listing the alternatives in successive lines within braces, and a special abbreviated notation for serial phrases is defined as:

$$\{A \parallel B\}^{+} \equiv A\{B\ A\}*$$

This is a generalisation of the method used by Woodward and Bond (1974). Now the user-oriented syntax of exiting conditionals can be compactly written as:

$$\text{IF } b \left\{\begin{Bmatrix}\text{THEN}\\ \text{CUT}\end{Bmatrix} sc \parallel \text{ELIF } b\right\}^{+} \left\{\text{ELSE } sc\right\}^{\phi} \text{FI}$$

where b stands for some clause yielding a Boolean, and sc is a serial clause. For implementation a more detailed syntax would be required, since the left and the right recursions have to be distinguished and, of course, no specification can guarantee the freedom from logical errors as it is always possible to block a section of a program by coding some tautological predicate, e.g. 'IF 2 * x * x > x * x CUT . . . FI'. This, however, occurs on a semantic level, and cannot be prevented.

A side-by-side example of a flow graph containing the exiting conditionals, and an equivalent skeleton program are shown in **Fig. 1.** The code should be largely self explanatory. Note that the introduction of CUT makes all but unnecessary the WHILE option in a loop, and in addition can be placed anywhere in it, not just at the head. For multiple exits from the same range one can discover which one actually took place by the value of some flag variable set just before the jump. Exits can be easily cascaded since the first statement after the exited closed clause can itself be another exiting conditional.

Lastly, we may note in passing that the abbreviated notation for series used here is a particular instance of a new hyper-rule definable by Van Wijngaarden syntax as:

NOTION1 separated by NOTION2 sequence:

NOTION1;

NOTION1 separated by NOTION2 sequence, NOTION2, NOTION1.

There are numerous other instances of such non-empty alternating chains used in definitions, so a compact notation applied here to the exiting conditionals could be quite useful in general.

References

VAN WIJNGAARDEN, A. *et al* (1976). *Revised report on the Algorithmic Language ALGOL 68*, Springer-Verlag.

WOODWARD, P. M. and BOND, S. G. (1974). *Algol 68-R Users Guide*, HMSO (Appendix 3).

Porting virtual object files

M. K. Crowe

Paisley College of Technology, Department of Mathematics and Computing, High Street, Paisley, Renfrewshire PA1 2BE

Software portability generally deals with moving source code from one machine to another. The development of 'virtual machines' emulated on a number of different systems opens the possibility of transferring binary object files.

In this paper, a method is described of overcoming the main snag that is encountered in transferring binary data from one machine to another. This is the fact that machines differ in word length, or even where the word length is the same, the way bits are arranged in a word.

The idea for this paper arose out of our work at Paisley in porting the EM-1 Pascal compiler developed at Vrije Universiteit, Amsterdam from the PDP-11 machine to our Prime computer.

The problem file

EM-1 is a stack based virtual machine architecture, with 16-bit word size. An executable object program file consists of

HEADER (WORDS)
TEXT OF PROGRAM (BYTES)
EXTERNAL DATA AREA
VARIOUS TABLES (WORDS)

The external data area consists of all program constants to be loaded into the stack prior to execution. This data is a mixture of bytes and words.

In the specification of the EM-1 machine, it is not defined how the bits in a word are arranged. Speed considerations demand that the same arrangement as in the host machine should be used. This is where the problem arose, as the PDP-11 and the Prime disagree on this point.

If two words, one of which contains the string

"AB"

and the second the integer 21, are transferred from the PDP-11 to the Prime, the resulting bit pattern is as follows

01 000 001 01 000 010 00 010 101 00 000 000

The Prime is fairly happy with the first word, which lacks only parity bits in the characters. But the bytes in the second word have been transposed. Similar problems will be encountered in transferring binary data from an Intel to a Motorola microcomputer.

Now there is no means of detecting this transposition when executing a program, as integers can be loaded using the same instruction as a two character string (loi 2). If the same object file is to be used, the only solution is to retain the PDP-11 ordering in the stack and swap bytes whenever any integer arithmetic requires to be performed. This must be ruled out because of the cost in execution time.

In our porting exercise, we took the usual way out of such difficulties, by porting ASCII files only. But in the future, as we begin porting on to various micros, some systematic way out of the difficulty will become essential.